## OptimusPi\_r1 Software Release Notes

WHAT HAS BEEN DONE IN THIS PROJECT ( all the work has been done only in the Overseer side )

Moved the code from Raspberry Pi down to Overseer -- done

including:

1) Added MPU9150 support, I2C/SMBus support on Overseer, and SPI commands for passing sensor data.

2) Added general-purpose timer on Overseer

3) Added printf support via JTAG

4) Added test environment for self-checking for peripheral drivers as well as debugging other codes

5) Added Arbitrator to switch fly mode between StandaloneCopter (controlled by RF controller)

or PiControlledCopter (controlled by RasbperryPi)

6) Fixed run-time bugs when code was moved from Raspberry Pi down to Overseer

@TODO New quadcopter prototype has not been built up yet.

@TODO New SPI commands and mechanism of switching fly mode has not been tested by using Raspberry Pi yet.

@TODO Standalone fly mode and PiControlledCopter fly mode has not been tested on hardware yet.

@TODO Logger class has been removed from control. Might be worth to added it back in for viewing

the status of the system and uploading it up to Raspberry Pi if necessary.

7) RX documentation. for pitch/roll/... kill switch documentation.

To add sw1/sw2 location and function description in rx.cpp

8) AHRS test code rename -- test code name too long.

WHAT IS LEFT TO DO:

1. On the Overseer side

Priority High:

1) To build up a quadcopter hardware prototype

i) Most of components can be found in the component list, which seats in the project directory

Picopter\_r1/documentations/optimuspi\_projects\_documents/OptimusPi\_r0\_documents/Created\_By\_Matthew/Quadcopter Component List.xlsx

Using the components provided above can assemble a hardware prototype

ii) To be noticed, when testing motors, each motor has power 100W with voltage 12V, i.e. rating current is 8.3A. Just four motors in the full power require about 33A. However, the maximum current of power supply in the lab currently is 18A, which is definitely not sufficient enough to drive all the four motors.

2) StandaloneCopterClass in src/TopLevel/StandaloneCopter/StandaloneCopter.cpp

i) To verify the quadcopter hardware

RF controller sw2 is the kill switch of the StandaloneCopter system.

3) RPiControlledCopterClass in src/TopLevel/RPiControlledCopter/RPiControlledCopter.cpp

i) To verify SPI mode-switching mechanism.

ii) To build a new update mechanism. Currently RPiControlledCopterClass uses an infinity loop to update system,

and the Overseer passes the raw data of sensor MPU9150 up to Raspberry Pi and Raspberry Pi is filtering noise itself.

In the future project, instead of using an infinity loop to update system, using a timer to call an ISR to update system

would give a finite system update rate, which would enable Overseer to filter sensor noise itself by using the Extended

Kalman Filter in the AHRS Class. (N.B. this mechanism has been implemented in the StandaloneCopterMode class,

which would be a good reference to build this mechanism in RPiControlledCopter Class) To be noticed, the update rate

required for AHRS class is normally between 100Hz and 400Hz (quoted from Matthew Watson), and therefore, the SPI

updating system in RPiControlledCopter needs to meet this time constrain. It is necessary to verify that the longest

updating cycle of RPiControlledCopter, i.e. RPiControlledCopterClass::running() method, is sorter than the fastest

AHRS update rate: 1/400Hz = 2.5ms.

iii)

Priority Midium:

2) more driver supports ported form sensorhub project.

Priorty Low:

3) TO free up SRAM.

i) only 32KB SRAM for run-time memory for the processor, and the processor currently uses 16KB for the heap(necessary for

printf support), 8KB for the stack(only less than 4KB of the stack size is currently required, just in case for future

projects, the stack size is set as 8KB). It is necessary to simplify some functional drivers to reduce the size of memory used.

3) src/I2C/I2C.cpp

i) Complete both SMBus and I2C support in I2C driver.

4) AHRSClass in

i) To build a simple In AHRS class, the Extended Kalman Filter uses a math matrix library Eigen. This math library is really costly in terms of CPU time. Reduction on the time running AHRS class per system update cycle can provide more bandwidth for other code to run. Furthermore, there is only 32KB SRAM for run-time memory in the processor

ii)

5) RX documentation update

i) More detail illustration of sw1 function which switch attitude control and rate control

2. On the Raspberry Pi side

1)